System Verification and Validation Plan for RoCam

Team #3, SpaceY
Zifan Si
Jianqing Liu
Mike Chen
Xiaotian Lou

October 23, 2025

Revision History

Date	Version	Notes
Date 1	1.0	Initial Draft

Contents

1 Symbols, Abbreviations, and Acronyms **iii**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 In-Scope Objectives . . . 1
- 2.4 Out-of-Scope Objectives . . . 2
- 2.5 Challenge Level and Extras . . . 2
- 2.6 Relevant Documentation . . . 3

3 Plan **3**
- 3.1 Verification and Validation Team . . . 3
- 3.2 SRS Verification . . . 3
- 3.3 Design Verification . . . 4
- 3.4 Verification and Validation Plan Verification . . . 7
- 3.5 Implementation Verification . . . 7
- 3.6 Automated Testing and Verification Tools . . . 8
- 3.7 Software Validation . . . 9

4 System Tests **10**
- 4.1 Tests for Functional Requirements . . . 10
  - 4.1.1 Area of Testing1 . . . 10
  - 4.1.2 Area of Testing2 . . . 11
- 4.2 Tests for Nonfunctional Requirements . . . 11
  - 4.2.1 Area of Testing1 . . . 11
  - 4.2.2 Area of Testing2 . . . 12
- 4.3 Traceability Between Test Cases and Requirements . . . 12

5 Unit Test Description **12**
- 5.1 Unit Testing Scope . . . 12
- 5.2 Tests for Functional Requirements . . . 12
  - 5.2.1 Module 1 . . . 12
  - 5.2.2 Module 2 . . . 12
- 5.3 Tests for Nonfunctional Requirements . . . 13
  - 5.3.1 Module ? . . . 13
  - 5.3.2 Module ? . . . 13
- 5.4 Traceability Between Test Cases and Modules . . . 13

6 Appendix **14**
- 6.1 Symbolic Parameters . . . 14
- 6.2 Usability Survey Questions? . . . 14

1 Symbols, Abbreviations, and Acronyms

symbol	description
Yolo	You only look once – an existing object detection model
CNN	Convolutional Neural Network
Jetson	abbreviatio for specifcally Jetson Nano Orin, which is the proposed board for this project
ST32	abbreviation for specifically ST32 microcontroller, which is the proposed microcontroller fo

This Verification and Validation (VnV) plan establishes the evidence that our RoCam model-rocket tracking system is both built right and the right build. Verification confirms each SRS and design requirement—camera and gimbal specs, timing/latency budgets, fault handling, and code quality—via reviews, static analysis, unit/integration tests, bench and simulation runs, with full requirement-to-test traceability. Validation demonstrates the system fulfills user and mission needs in field conditions by acquiring the rocket at launch, maintaining continuous tracking through boost/coast and occlusions, and supporting a safe, simple operator workflow. Success is judged against quantitative targets and operational criteria. Deliverables include test plans/cases, automated reports, calibration and operating procedures, field-test results, and a validation summary with stakeholder sign-off.

TODO: tidy up deliverables list

2 General Information

2.1 Summary

The RoCam system is a ground-based camera and gimbal assembly designed to autonomously track model rockets during launch and flight. It utilizes a high-resolution camera coupled with a motorized gimbal to maintain visual lock on the rocket, providing real-time data and video feed. The system consists of a STM32 based motion controller that interfaces with the physical gimbal; a Jetson compute module that processes the camera feed; and a react frontend that displays the video preview and allows the user to control the gimbal. The primary purpose of RoCam is to replace the traditional unreliable manual tracking methods with an automated solution to produce high-quality flight data and video footage. To better assist model rocket teams across Canada to conduct successful launches by providing an affordable and reliable system that can be easily track launching movements.

2.2 Objectives

The primary objective of the system is to **demonstrate reliable, real-time visual tracking of small-scale model rockets** through an autonomous, vision-guided camera system. The project aims to achieve **high accuracy**, **low latency**, and **stable video performance**, ensuring that the rocket remains centered in frame throughout its flight. By focusing on **system performance**, **tracking precision**, and **robustness under real launch conditions**, RoCam seeks to provide high-quality flight footage that supports post-launch analysis and safety validation.

2.3 In-Scope Objectives

Verification and validation efforts will prioritize the following qualities:

- **Correctness:** Ensure the reliability of tracking and control algorithms through testing and validation.
- **Performance:** Achieve real-time processing of 1080p 60 fps video and maintain minimal gimbal control latency.

- **Reliability and Safety:** Guarantee that the system operates s afely and consistently under outdoor launch conditions.

2.4 Out-of-Scope Objectives

The following objectives are excluded from this project due to time, budget, and resource constraints:

- **Comprehensive Usability Testing:** User experience will be evaluated informally via team and client feedback rather than large-scale formal studies.
- **Third-Party Component Verification:** External hardware such as the gimbal controller and camera are assumed to perform according to their manufacturer specifications.
- **Scalability Beyond Small-Scale Rockets:** Tracking a bullet or extremely difficult targets beyond a model rocket is outside the scope of this project.

By defining these priorities and exclusions, the RoCam project ensures that available effort is directed toward validating the most critical system qualities: **performance, accuracy, and operational safety**.

2.5 Challenge Level and Extras

The **RoCam** project is an engineering heavy capstone project. One of the major challenges of this project is the integration of multiple complex subsystems, including **real-time computer vision**, **hardware-software interfacing**, and **autonomous motion control**. The project requires the implementation of algorithms capable of detecting, tracking, and maintaining a visual lock on fast-moving model rockets, while ensuring real-time performance at 1080p 60 fps and stable gimbal control under dynamic outdoor conditions. In addition, the system's design encompasses both software engineering and embedded hardware domains, further contributing to its advanced technical scope.

Extras

Beyond the core objectives, the RoCam project includes several **approved extras** that enhance its quality and usability:

- **STM32 control circuit design:** Development of a custom control circuit to interface the gimbal actuators with the host computer, enabling precise motion control and feedback sensing.
- **User Instructional Video:** Creation of a short instructional video demonstrating system setup, calibration, and operation to support new users and ensure safe deployment.
- **Usability Evaluation:** Informal field-based usability evaluation with feedback from the McMaster Rocketry Team to validate ease of use and interface clarity.

These extras aim to improve the overall usability, accessibility, and documentation quality of the final deliverable, reinforcing RoCam's emphasis on practical integration between computer vision software, mechanical control, and end-user experience.

2.6 Relevant Documentation

IDK What to do here
Author (2019)

3 Plan

3.1 Verification and Validation Team

3.2 SRS Verification

The verification and validation (V&V) process for the **RoCam** project is carried out collaboratively by all team members under the supervision of the faculty advisor. Each member is responsible for verifying specific components of the system based on their primary area of contribution, ensuring that both software and hardware subsystems meet performance, safety, and usability requirements. Table 1 summarizes the V&V responsibilities for each team member.

Table 1: Verification and Validation Team Responsibilities

Name	Role	V&V Responsibilities
Zifan Si	Team member	Responsible for verifying and validating the usability and functionality of the react frontend interface. Ensure usability, robustness of frontend features.
Jianqing Liu	Team Lead	Conduct integration test and system validation. Validate the functionality and performance of the system against requirements and use cases.
Mike Chen	Hardware and Control Lead	Verify and validate computer vision pipeline performance, including object detection and tracking accuracy.
Xiaotian Lou	Software Quality and Testing Lead	Implement automated testing frameworks in all software components.
Dr. Shahin Sirouspour	Faculty Supervisor	Provides oversight and ensures that verification and validation procedures align with academic and engineering standards. Reviews test methodologies, validates safety-critical procedures, and confirms that deliverables meet course and institutional requirements.
Launch Canada	Stakeholder	Provides feedback on system performance during field tests.

Together, this team structure ensures that verification and validation activities are distributed across all critical system components—computer vision, control, integration, and software reliability—resulting in a robust and well-tested final product.

3.3 Design Verification

The design verification process for the **RoCam** project ensures that the proposed architecture, algorithms, and hardware interfaces satisfy all functional and non-functional requirements before implementation. Design verification activities will be conducted through structured design reviews, peer evaluations, and the use of detailed verification checklists. The goal is to confirm that the design meets the intended performance, reliability, and safety objectives defined in the SRS.

Verification Plan

The design verification will proceed in three main stages:

1. **Internal Design Review:** Each subsystem (computer vision, control, and UI) will undergo an internal review by each team member. Each member must agree on the

designed architecture, algorithms, and interfaces. Interfaces must be clearly defined for scalability.

2. **Peer Review by Classmates:** A peer design review session will be conducted with other capstone teams to obtain external feedback. Reviewers will evaluate the system's clarity, feasibility, and maintainability based on the shared design artifacts (block diagrams , interface specifications, and test plans).

3. **Supervisor and TA Review:** The project supervisor and teaching assistants will review the finalized design to ensure compliance with academic standards, capstone expectations, and safety considerations.

Verification Artifacts

Design verification will produce the following artifacts:

- Verified design diagrams and interface specifications.
- Annotated review feedback forms from classmates and supervisor.
- Updated design documents reflecting corrections and improvements.
- Completed design verification checklists (see below).

Design Verification Checklist

The following checklist will be used during design reviews to ensure completeness and consistency across all modules:

Table 2: RoCam Design Verification Checklist

Verification Item	**Status (✓/✗)**
Requirements Traceability: Each design module (vision, control, UI, data management) maps directly to specific functional and non-functional requirements in the SRS.	
System Architecture Validation: The proposed architecture supports concurrent data acquisition, vision inference, and gimbal control without violating latency constraints.	
Performance Objectives: Design supports 1080p 60 fps video input/output, with an end-to-end latency target of less than 120 ms between camera frame and gimbal response.	
Rocket Tracking Algorithm: The computer vision subsystem's design ensures accurate rocket detection, tracking, and reacquisition under changing lighting and smoke conditions.	
Gimbal Control and Stability: The control architecture provides smooth, stable movement and sufficient angular speed for small rocket tracking.	
Hardware–Software Interface: Communication between the Jetson platform, STM32 controller, and gimbal is well-defined, with verified protocol timing and error handling.	
Safety and Fault Handling: Safe-state transitions (Idle, Armed, Tracking) are fully defined and prevent uncontrolled gimbal motion on error or manual override.	
Reliability and Robustness: Design includes failsafes for connection loss, power interruption, and unexpected frame drop events during operation.	
User Interface Design: The UI supports outdoor readability, bilingual support (English/French), and visual status indicators for all operational states.	
Data Recording and Storage: Recording subsystem design supports synchronized video and gimbal-angle logging at 1080p 60 fps, with enough local storage for a full launch day.	
Testing and Validation Plan Alignment: Each subsystem has a defined verification procedure (unit tests, simulation, or field test) linked to corresponding SRS fit criteria.	
Peer Review and Feedback Resolution: All feedback from peer design reviews and supervisor evaluations has been documented and addressed in the updated design artifacts.	

By following this structured verification plan and checklist, the team ensures that the RoCam design is both technically sound and fully aligned with its performance and safety goals prior to implementation.

3.4 Verification and Validation Plan Verification

3.5 Implementation Verification

The implementation verification for the **RoCam** project ensures that the developed software and integrated hardware components correctly realize the verified design. This phase confirms that the implemented system satisfies the functional and non-functional requirements defined in the SRS through a combination of structured testing, static verification, and peer review.

Dynamic Verification

Dynamic verification activities focus on confirming that the system behaves as intended under realistic operating conditions. They include the following elements:

- **Unit Testing:** Each software module (vision processing, gimbal control, data recording, and UI management) will undergo unit testing to verify functional correctness in isolation.
- **Integration Testing:** Once individual modules are verified, integration tests will confirm proper data flow between subsystems (camera input $\rightarrow$ vision algorithm $\rightarrow$ gimbal control $\rightarrow$ UI). Tests will ensure synchronization, timing correctness, and robustness under real-time constraints.
- **System Testing:** Full-system field tests will validate end-to-end performance, including real-time video tracking, system state transitions (Idle, Armed, Tracking), and video recording accuracy at 1080p 60 fps. These tests correspond directly to the verification items in the SRS functional and performance requirements.

Static Verification

Static verification will be performed throughout the implementation phase to ensure that the codebase meets quality, safety, nd maintainability standards before execution. Planned techniques include:

- **Code Walkthroughs:** Conducted during the final CAS 741 presentation, where each subsystem lead presents core implementation logic and design decisions. This serves as a semi-formal walkthrough for peer and instructor review, enabling early identification of potential design or coding issues.
- **Code Inspection:** Team members will review each other's pull requests prior to merging into the main branch. Reviews will focus on coding standard compliance, potential logic errors, exception handling, and safety-critical sections (e.g., gimbal actuation and state transitions).

- **Static Analysis Tools:** Tools such as `clang-tidy`, `cppcheck`, and `pylint` will be used to detect memory leaks, uninitialized variables, and other common implementation defects. Results from static analysis will be logged and reviewed as part of the verification record.

- **Documentation Review:** The team will verify that all inline comments, module-level documentation, and API references are consistent with the design specification and user documentation requirements.

Presentation and Usability Review

The final CAS 741 presentation will serve as both a **code walkthrough** and a partial **usability evaluation**. During the presentation, peers and instructors will observe live system operation, assess user interface clarity, and provide feedback on usability, responsiveness, and overall system behavior. This qualitative feedback will be incorporated into the final verification summary and used to refine user-facing components prior to final submission. Together, these verification activities ensure that the RoCam implementation is thoroughly tested, statically verified, and validated against both its design and real-world performance expectations.

3.6 Automated Testing and Verification Tools

For Phase 1, the implementation is **Python-first** (OpenCV pipeline, control logic, and operator UI services). Our automation focuses on Python and the web UI; any C/C++ tools will be introduced only in a later phase if needed.

Unit and Integration Testing

- **Python:** `pytest` with `pytest-cov`; fixtures simulate camera frames (OpenCV `VideoCapture` stubs), gimbal feedback, and UI requests.

- **Web UI (if applicable):** Jest + Testing Library for component tests; Playwright for smoke-level end-to-end checks covering the operator workflow (arm → track → stop → export).

Static Analysis, Linters, and Style

- **Python:** `flake8` and `pylint` (lint), `mypy` (type checking for critical paths), `bandit` (basic security checks).

- **Formatting:** `black` + `isort`; enforced via pre-commit hooks and CI.

- **Web UI:** ESLint with project rules; Prettier for formatting.

Coverage and Build Automation

- **Coverage (Python):** `coverage.py` via `pytest-cov`; HTML reports published by CI.
- **Coverage (Web UI):** Istanbul/NYC for statements/branches/functions/lines.
- **Build/Tasks:** `pip` + `pyproject.toml/requirements.txt`; `npm` scripts for the UI.
- **CI/CD:** GitLab CI pipelines run on every push/MR; jobs enforce: lints clean, type check passes, unit tests pass, and minimum coverage gates.

Profiling and Runtime Diagnostics

- **Python:** `cProfile`, `line_profiler`, and `memory_profiler` on CV hot paths; timestamped logs to measure camera→inference→gimbal command latency.
- **Observability:** Structured logs (JSON) with frame IDs and timing; optional Prometheus-style counters for latency histograms (median/95th).

Coverage Reporting Plan

- **Thresholds:** Line coverage $\geq 80\%$ for safety-critical modules (state machine, tracking loop, command arbitration); $\geq 70\%$ for other modules. Branch coverage is reported for state-transition logic.
- **Reports:** CI publishes HTML artifacts and a coverage badge; weekly trend tracked and cited in the V&V report.
- **Scope Notes:** Hardware I/O shims (e.g., serial/gimbal stubs) are measured separately; excluded/generated code is documented.

3.7 Software Validation

Software validation confirms that **RoCam** satisfies stakeholder needs and accurately implements the intended use cases ("building the right system"). Validation complements verification and is referenced to the SRS goals nd stakeholder roles.

Stakeholder Reviews and Task-Based Inspection

- **Requirements Walkthroughs:** Scheduled reviews of the SRS and UI wireframes with the McMaster Rocketry Team (client) to confirm scope, priorities, and acceptance criteria.
- **Task-Based Inspection:** Stakeholders perform representative tasks (arm system, initiate tracking, stop tracking, export footage) while observers record issues, ambiguities, and unmet expectations.
- **Peer Reviews:** Classmate reviews focus on clarity, testability, and completeness (per CAS 741 guidance); all feedback and resolutions are logged.

Field and Demo-Based Validation

- **Rev 0/PoC Demo:** Used to validate that early functionality aligns with stakeholder expectations (e.g., preview latency, arm/track state semantics). Post-demo debrief captures change requests.
- **Supervisor Validation:** A focused session shortly after Rev 0 to confirm that the demonstrable behaviors match the SRS's product use cases and performance intent.
- **User (Operator) Sessions:** "Hallway" usability checks with camera operators validate readability in outdoor-like conditions and UI flow (single-page ops without scrolling on 1920×1080).

External Data and Bench Validation

- **Reference Footage:** Where live launches are unavailable, public model-rocket videos and internally recorded dry-runs are used to validate detection/tracking logic and reacquisition behavior.
- **Synthetic Scenarios:** Scripted clips with occlusion/smoke/backlight and varying apparent rocket sizes validate robustness claims prior to field tests.

Acceptance and Traceability

- **Acceptance Criteria:** Derived from the SRS (e.g., sustained 1080p 60 fps I/O, end-to-end latency ≤ 120 ms, successful tracking through nominal flight phases, recording with a ngle overlays).
- **Traceability:** Validation tests map to SRS goals and stakeholder needs; a table links each validation activity to the corresponding SRS item(s) and planned evidence (videos, logs, checklists).

If no suitable external datasets are available for certain scenarios, this limitation will be stated explicitly; in those cases, validation will rely on stakeholder task-based inspections and controlled synthetic footage. This section also references the SRS verification section for consistency checks between validated behaviors and specified requirements.

4 System Tests

4.1 Tests for Functional Requirements

4.1.1 Area of Testing1

Title for Test

1. test-id1

Control: Manual versus Automatic

Initial State:

Input:

Output:

Test Case Derivation:

How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

5 Unit Test Description

5.1 Unit Testing Scope

5.2 Tests for Functional Requirements

5.2.1 Module 1

1. test-id1

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

2. test-id2

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

5.3.1 Module ?

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?